2004

Data mining and data warehousing using granular computing

Hooman Moobed
San Jose State University

DATA MINING AND DATA WAREHOUSING USING GRANULAR COMPUTING

A Thesis

Presented To

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

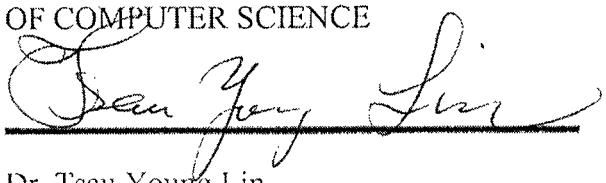
by

Hooman Moobed

August 2004

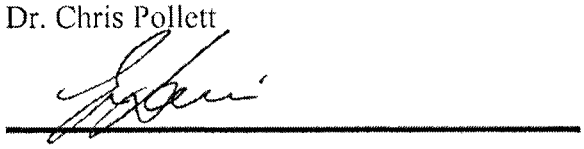
APPROVED FOR THE DEPARTMENT

OF COMPUTER SCIENCE



Dr. Tsau Young Lin

Dr. Chris Pollett



Eric Louie

[IBM ALMADEN RESEARCH CENTER]

APRROVED FOR THE UNIVERSITY

ABSTRACT

DATA MINING AND DATA WAREHOUSING USING GRANULAR COMPUTING

by Hooman Moobed

This thesis explains and compares the following three granule-based computing techniques: Granule Based Appriori, Granule Based Appriori Lattice, and Granule Based Lattice. The latter two algorithms are new and are proposed in this paper. This thesis also explains a technique to store the data in a granule form in a data warehouse.

A granule is a compact list of tuple names having the same attribute value and is represented in binary form. With the data converted to granules, the relationships are determined by the intersection of the granules. The operations performed on the granules are natural instructions for the computer. This has a great significance in the speed of computation. It is costly and time consuming to access and process the data to find patterns. Thus, it is important for the storage system to not only provide fast response time but also provide the data in a format most suitable for processing.

Table of Contents

List of tables

List of figures

1 Introduction

It is estimated that the amount of information in the world doubles every 20 months [1]; that is, many scientific, government, and corporate information systems are being overwhelmed by a flood of data that are generated and stored routinely, which grow into large databases. These databases contain potential gold mines of valuable information, but it is beyond human ability to analyze such massive amounts of data and elicit meaningful patterns. Data mining is a possible way to interpret these data.

Data mining refers to extracting or mining of knowledge from large amounts of data. It involves an integration of techniques from multiple disciplines such as database technology, high-performance computing, pattern recognition, etc. By performing data mining, interesting knowledge, regularities, or high-level information can be extracted from databases and viewed or browsed from different angles. The discovered knowledge can be applied to decision-making, process control, information management, and query processing.

One area in data mining is the discovery of association rules or patterns in the data. Rakesh Agrawal introduced the Appriori method, in the early 1990s, to find association rules. Association rule mining finds associations or correlation relationships among a large set of data items. The discovery of patterns among huge amounts of business transaction records can help in many business decision-making processes, including catalog design, cross marketing, and loss-leader analysis. A typical example of

association rule mining is market basket analysis. This process analyzes customers' buying habits by finding associations between the different items that customers place in their shopping baskets. The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely they are to also buy bread and what kind of bread are they likely to buy on the same trip to the supermarket. Such information can lead to increased sales by helping retailers to do selective marketing and to plan their shelf space. For example, placing milk and bread within close proximity may further encourage the sale of these items together within single visits to the store.

Discovering the semantics of the data is a computational task, requiring many reads of the data and many comparisons of attribute values within the data. In addition, the organization of the data is known to have a huge effect on how quickly an answer is produced. The data model that uses granules to represent the attribute values offers another approach to find semantics from the data. This thesis paper explains the concepts like universe, entities and granules. It explains how this data model is represented, how it is stored in a data warehouse and how it is used to discover association rules from the data. Lastly, this thesis explains and compares the following three granule-based computing techniques: Granule Based Appriori, Granule Based Appriori Lattice, and Granule Based Lattice. The latter two algorithms are new and are proposed in this paper.

2 Data Representation

The universe contains entities and each entity in the universe has properties [5]. This universe is the knowledge representation of real world entities. The universe is the relation in the relational database theory. Each entity in the universe represents one tuple in the relation and each tuple represents one entity in the universe. The properties of the entities are the attribute values in the relation. Each property induces a partition in the universe. The partition is a subset of the entities in the universe. The entities with the same properties are grouped together, thus partitioning the universe. Each of these partitions is a mutually disjoint equivalence class.

Pawlak's Rough Set Theory describes the relationship between the mathematical term, Universe (V) and Equivalence Class (E), in the following figure:

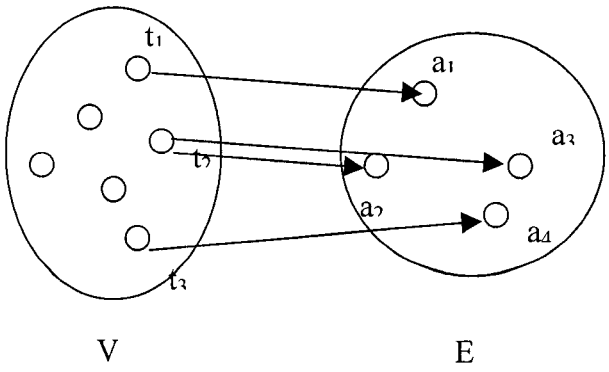


Figure 1. Rough Set Theory (Tuples pointing to attributes)

The Rough Set Theory is a natural generalization of the "twin" theory (well known in interval mathematics). In both theories, we are interested in a set S, which can be a set of possible values of some quantity or a set of pixels that form an image.

According to Pawlak's theory there are two ways to represent a tuple. In the first case the tuple points to the attributes it belongs to. In the second case the attributes point to the tuples they belong to.

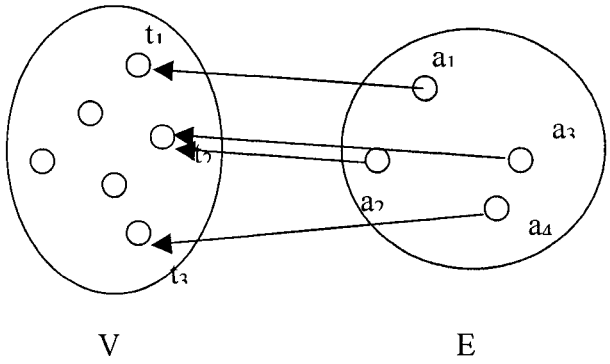


Figure 2. Rough Set Theory (Attributes pointing to tuples)

The relations between the Universe (V) and the Equivalence class (E) are many-to-many.

The equivalence relations are the granules of the relation. Each unique attribute value in the relation is a granule and each granule is the list of tuples that have the same attribute value. Only the tuple name or the reference to the tuple is stored in the granules. Other than a list, the granules can be represented in bit streams. Each tuple in the relation has a unique offset position in the bit stream. The bit is set to one for those tuples that have the attribute value of the granule and are set to zero for those that do not have the attribute value of the granule.

A granule is a compact list of tuple names having the same attribute value and it is represented in a binary form. With the data converted to granules, the relationships are determined by intersection of the granules [2]. This has a great significance in the speed of computation.

2.1 Granule Generation

These concepts are applied below to a simple relation. A mapping f: X -> Y is mathematically defined as a single valued function, f, which assigns to each element of X a unique value of Y. In this case, f = K, X = V and Y = Dom(S#) x Dom(STATUS) x Dom(CITY).

V	K	S#	SNAME	STATUS	CITY
V1	->	S1	Smith	Twenty	C1
V2	->	S2	Jones	Ten	C2
V3	->	S3	Blake	Ten	C2
V4	->	S4	Clark	Twenty	C1
V5	->	S5	Adams	Thirty	C3

Table 1 Simple Relation

For the sake of brevity, only columns S# and CITY from the above table will be discussed. S# represents the ID or each person, which is unique among all the tuples in the relation. CITY represents the city that the person is from. The values for this attribute are not unique. The quotient set for the CITY attribute is {C1, C2, C3}. Each element in the quotient set represents the name of a granule and is the equivalence class that partitions the above relation. The granules are shown below in two forms, the list form and the binary form.

City quotient set in list form

C1 = {S1, S4}

C2 = {S2, S3}

C3 = {S5}

City quotient set in binary form

C1 = {10010}

C2 = {01100}

C3 = {00001}

The list form has the values from S#. The advantages of this form are ease of readability and identification of the tuples in the relation. The binary form of the granules contains a bit stream of zeroes and ones with the offset positions marking the tuples with a particular attribute value. In either form they represent the granules in the relation.

Converting the city relation to binary form can be done in the following four steps:

1) Take each value in the quotient set and create a new attribute in a new table.

S#	C1	C2	C3

2) Mark the appropriate attribute in the new table for each tuples attribute value in the old table. A value of "1" is placed on the attribute in the new table that is the

matching attribute value of that tuple in old table. A value of '0' is placed on the other attributes that do not match the attribute value of the tuple from the old table.

S#	C1	C2	C3
S1	1	0	0
S2	0	1	0
S3	0	1	0
S4	1	0	0
S5	0	0	1

3) Rotate the above table 90 degrees.

S#	S1	S2	S3	S4	S5
C3	0	0	0	0	1
C2	0	1	1	0	0
C1	1	0	0	1	0

4) Take all the values in the rows and compress them into a binary stream of zeroes and ones.

Quotient Set	Binary form
C3	{00001}
C2	{01100}
C1	{10010}

The three binary representations are the canonical names of each granule. For a single column relation, the union of the three binary representations is the relation and the intersection of the three binary representations is empty. Therefore the granules are disjoint subsets of the relation. The number of zero bits in the granules increases with the cardinality. The number of 1 bits increases with the number of rows. Hence, as the cardinality increases with the same number of rows, more and more zero bits occur with

the same number of one bits. The granules are used to find the relationships between attribute values. This is called granular computing.

2.2 Granule Computation

A combination or candidate consists of one or more attribute values. As granules, a combination is an association rule if the bit count on the intersection of all the granules is equal to or greater than the minimal count of that relation. The bit count is the number of 1's in the bit stream from the result of the intersection of one or more granules. The bit representation of the granules offers efficient storage while the intersection of the granules offers fast computation in finding the association rules.

Let's consider the STATUS and CITY columns from Table 1.

STATUS	CITY
Twenty	C1
Ten	C2
Ten	C2
Twenty	C1
Thirty	C3

The quotient set and the binary stream representation for each of the above columns is as follows:

Quotient Set (STATUS)	Binary form
Ten	{01100}
Twenty	{10010}
Thirty	{00001}
Quotient Set (CITY)	Binary form
C3	{00001}
C2	{01100}
C1	{10010}

From the above two tables, one can conjecture that the STATUS of Twenty and the CITY C1 appear 40% of the time in our data. This conjecture can be verified with the intersection of two granules, i.e. the bit representation of the value Twenty from the STATUS attribute, and the value C1 from the CITY attribute.

$10010 \cap 10010 = 10010$

After the intersection, the result is counted. In this case the bit count that is the number of 1's in the bit stream from the result of the intersection from the above two granules is 2. This bit count is out of the total of 5 records or bits in our example. The following calculation shows that the STATUS of Twenty and the CITY C1 appear 40% of the time in our data. (2 / 5) * 100% = 40%.

2.3 Granule Bit Count

If the granules are represented in bit form, the intersection of granules in bit form requires the bit-wise AND operation among the granules. The bit-wise operation is done using the machine's word size operations. The AND operation is done to find the intersection between the granules. After the AND operation, the bits in the result are counted. The bits (that are set to 1) need to be counted to determine if the pattern is an association rule. The pattern becomes a rule if the intersection of the granules results in a count that exceeds the minimum count of that relation.

Below is a method of counting the bits in the result of an intersection. Assuming the machines word size is 32 bits, we would count 32 bits of the intersection at a time.

The bits are counted by repeatedly dividing the word or subtotal by 2^x and adding the first half with second half on each division. The letter, x, begins with "1" and increments on each cycle until 2^x equals the length of the subtotal. The bit count of the result is done one word size at a time. First, each pair of bits in a word is counted by adding the even bits with the odd bits. This step has 16 subtotals. Then, every adjacent pair of the subtotals is added together, resulting in half the number of subtotals from the previous step. This step is repeated until the number of subtotals results in one total. The complexity of this algorithm is O (log b) where b is the number of bits in a granule.

Let's assume the following binary representation is 32 bits of the result of our intersection for which the set bits need to be counted:

0011 1100 0101 1110 0000 1111 0111 1001

Step 1: Each pair of bits in a word is counted by the addition of the even bits to the odd bits.

Even bits:

0x 1x 1x 0x 0x 0x 1x 1x 0x 0x 1x 1x 0x 1x 1x 0x

Odd bits:

x0 x1 x1 x0 x1 x1 x1 x0 x0 x0 x1 x1 x1 x1 x0 x1

This step has 16 subtotals:

00 10 10 00 01 01 10 01 00 00 10 10 01 10 01 01

Step 2: Each pair of 4 bits in the above subtotal are counted by adding the even 2 bits with the odd 2 bits.

Even 2 bits:

00xx 10xx 01xx 10xx 00xx 10xx 01xx 01xx

Odd 2 bits:

xx10 xx00 xx01 xx01 xx00 xx10 xx10 xx01

This step has 8 subtotals:

0010 0010 0010 0011 0000 0100 0011 0010

Step 3: Each pair of 8 bits in the above subtotal are counted by adding the even 4 bits with the odd 4 bits.

Even 4 bits:

0010xxxx 0010xxxx 0000xxxx 0011xxxx

Odd 4 bits:

xxxx0010 xxxx0011 xxxx0100 xxxx0010

This step has 4 subtotals:

00000100 00000101 00000100 00000101

Step 4: Each pair of 16 bits in the above subtotal are counted by adding the even 8 bits with the odd 8 bits.

Even 8 bits:

00000100xxxxxxxx 00000100xxxxxxxx

Odd 8 bits:

xxxxxxxx00000101 xxxxxxxx00000101

This step has 2 subtotals:

0000000000001001 0000000000001001

Step 5: The 32 bits in the above subtotal are counted by adding the even 16 bits with the odd 16 bits.

Even 16 bits:

0000000000001001 xxxxxxxxxxxxxxxx

Odd 16 bits:

xxxxxxxxxxxxxxxx 0000000000001001

This step has the subtotal of:

00000000000000000000000000010010

The result is a binary value, 18, which is the number of bits set to 1 in our 32 bit binary representation.

The sample 'C' code to count the bit sets is described below:

```
unsigned long bits;
unsigned long count;
unsigned long total;
```

```
count = (bits    & 0x55555555)  + ((bits    >> 1) & 0x55555555);
count = (count & 0x33333333)  + ((count >> 2) & 0x33333333);
count = (count & 0x0F0F0F0F) + ((count >> 4) & 0x0F0F0F0F);
count = (count & 0x00FF00FF) + ((count >> 8) & 0x00FF00FF);
total += (count & 0x0000FFFF) + (count >> 16);
```

Here is an explanation for the above code segment:

Step 1: count = (bits & 0x55555555) + ((bits >> 1) & 0x55555555);

The first part of the above expression (bits & 0x55555555) gets the odd bits that are set. The next part ((bits >> 1) & 0x55555555) gets the even bits that are set and then we obtain the subtotal by adding the two.

Step 2: count = (count & 0x33333333) + ((count >> 2) & 0x33333333);

The first part of the above expression (count & 0x33333333) gets the odd 2 bits that are set. The next part ((count >> 2) & 0x33333333) gets the even 2 bits that are set and then we obtain the subtotal by adding the two.

Step 3: count = (count & 0x0F0F0F0F) + ((count >> 4) & 0x0F0F0F0F);

The first part of the above expression (count & 0x0F0F0F0F) gets the odd 4 bits that are set. The next part ((count >> 4) & 0x0F0F0F0F) gets the even 4 bits that are set and then we obtain the subtotal by adding the two.

Step 4: count = (count & 0x00FF00FF) + ((count >> 8) & 0x00FF00FF);

The first part of the above expression (count & 0x00FF00FF) gets the odd 8 bits that are set. The next part ((count >> 8) & 0x00FF00FF) gets the even 8 bits that are set and then we obtain the subtotal by adding the two.

Step 5: total += (count & 0x0000FFFF) + (count >> 16);

The first part of the above expression (count & 0x0000FFFF) gets the odd 16 bits that are set. The next part (count >> 16) gets the even 16 bits that are set and then we obtain the subtotal by adding the two.

3 Data Warehousing

The organization of the data has a huge impact on how quickly a pattern is found. A data model that uses granules to represent the data offers an alternate approach to finding patterns within the data. The major significance of a data model is its computational speed.

The use of indexes to accelerate query processing has long been used in all RDBMS (Relational Databases Management System) products. When various types of indexes are defined on selected columns of a table, and the query constraints on those columns are defined, the RDBMS can use those indexes to quickly identify the relevant rows that satisfy the selection constraints. Most traditional databases use a balanced-tree (B-Tree) index strategy for rapid retrieval [4]. This means that they can find a small number of data values quickly by traversing the tree to reach the relevant pages of data. A B-Tree index keeps track of values of selected fields and points directly to data pages that contain them. A B-Tree index is valuable only if there are many data values that are used to filter down to a small set of records. Columns with only a few unique values, such as gender or marital status, are not good candidates. In a data mining application, highly selective queries are not the norm. The opposite is true, so the previously described indexing techniques do not work very well for data mining applications. For a database kernel to optimize the data mining process, it must have other indexing techniques at its disposal. The granule based indexing technique is one such approach.

The basic algorithm used in our data warehouse to convert records in the database to granules is as follows:

1) Analyze the data.
2) Read the data.
3) Generate granules.
4) Calculate the layout position of the granules.
5) Write the granules to disk.

The flow chart of this algorithm is depicted below:

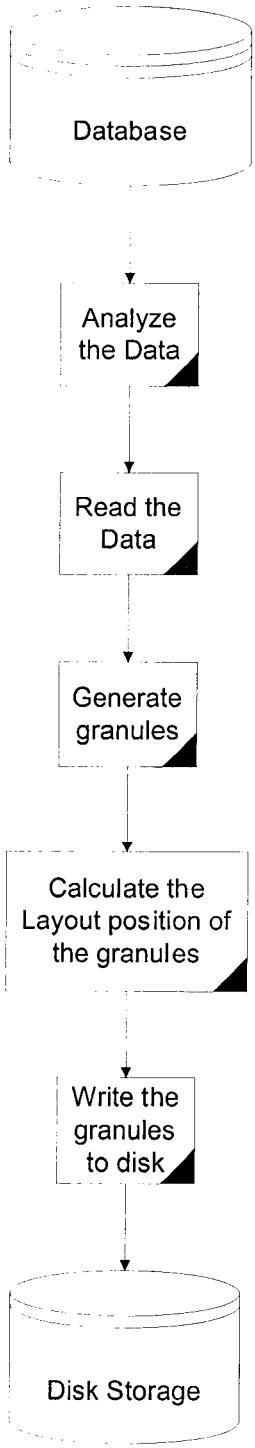


Figure 3. Flow chart to convert database records into granules

3.1 Data Warehouse Components

Our granule based data warehouse is made up of multiple components. The data access component connects to a specified database and analyzes the data. It then converts the data into granules and passes the granules to the data layout component. The data layout component controls how the granules are positioned within the data storage component. It then passes the granules to the storage component and updates the indexer of their positions. Each of these components is described in more detail below.

3.1.1 Storage Component

The storage component is based on the VSAM file system. Once the data is converted into granules it is stored on the disk using this component. The storage component is characterized by data blocks (control areas and control intervals) with no necessary physical contiguity, and a threading of index blocks at the level immediately above the data (sequence set). The upper levels, down to the sequence set, are managed by the indexer component. The non-contiguity of data blocks allows for graceful handling of non-uniform distributions of inserts. The threading above data level allows efficient sequential and range access. New memory is allocated in contiguous control areas, divided into a number of control intervals.

3.1.1.1 Control Interval

With conventional access methods, the unit of data that is moved between virtual storage and direct access storage is called a block. With data warehouse, the unit of data that is transferred in each physical I/O operation is called a control interval. A control interval contains section(s) of a granule(s), control information, and possibly free space.

Figure 4. Control Interval

The data format of our Data warehouse supports fixed-length records. The records are mapped into fixed-length Direct Access Storage Device (DASD) read and write units called *Control Intervals* (CIs). Control information resides at the end of each control interval. There are two sets of control information. The last four bytes of the CI is the Control Interval Definition Field (CIDF). Immediately to the left of the CIDF is one or more record definition fields (RDFs). An RDF describes one record, a set of contiguous fixed-length records, or a segment of a spanned record. A spanned record has a length greater than the size of a control interval, and thus it resides in a set of control intervals. Our control intervals contain fixed length records and have two RDFs, regardless of the number of granule sections stored in them. The first RDF contains the record length. The second RDF contains the number of currently stored records in the control interval.

The control interval architecture includes a distributed free space capability. Records are stored within a control interval starting at offset zero. The RDFs specify the lengths of each record and are stored in reverse order beginning immediately to the left of

the CIDF. Any unused space between the end of the last record in the CI and the last RDF is free space. The basic data warehouse data addressing model is a linear space model. The term Relative Byte Address (RBA) is used to designate the address of a data location within a data warehouse linear space. Although the byte address technique is used, the data warehouse data are stored and accessed as records.

When the information is requested from or supplied to the data warehouse, the unit of information transferred is a logical record. Data warehouse uses a form of blocking called a control interval to contain records and control information. When a record is retrieved, the entire control interval is read into a Data warehouse I/O buffer in the virtual storage. Then the desired record is transferred to a buffer or work area for the program to use. If the next record to be read is in the same CI then another physical read is not necessary as the record is in the buffer. CIs are further blocked into control areas that vary in size from 1 track to 1 cylinder.

3.1.1.2 Control Area

Control intervals are grouped together into control areas. As the data warehouse is loaded, control areas are created and control intervals are written into them. Control areas are filled with control intervals that contain records, as required. Some of the control intervals in each control area consist entirely of free space that can be used for data set expansion.

A Control Area (CA) is a set of contiguous control intervals. This area defines a two-level space hierarchy within the linear space of the data component of the data

warehouse. For each control area of the data component, data warehouse assigns a control interval of the index component's linear space. This index control interval is called a *sequence set control interval.* The sequence set control interval contains one entry for each data control interval within the corresponding data control area.

A data control interval may contain data records or, it may be empty. An empty data control interval is called a *free space control interval.* The index sequence set control interval of the data control area contains two sets of information. One set is a key-ordered index containing a compressed key value along with a data control interval pointer for each data control interval that contains data records. The other set is a list of free space data control intervals.

The free space is maintained within each data control interval and a set of empty data control intervals within each control area. As records are inserted, their storage position is determined by the index and the existing records within the data set. When a record is inserted within a data control interval, any existing data records within that control interval with higher key values are shifted to make room for the new record.

When an insertion occurs without sufficient free space within the data control interval, the data control interval is split. The split consists of allocating a free space control interval from the free list within the sequence set control interval of the control area, and distributing records across the two data control intervals and inserting the record. When there is not a free space data control interval within the data control area, a control area split is performed. This split allocates a new control area from the end of the data set and distributes the data across the two control areas, forming free space control

intervals within each of the two control areas. When a control interval or control area split occurs, the index is updated to reflect the split.

3.1.2 Indexer Component

The indexer component of the data warehouse maintains a list of key values with pointers to the data within the storage component. The data warehouse uses the indexer component to locate the granules stored in the storage component for retrieval. Depending on the layout of the granules in the storage component, they can be accessed sequentially, in order, or directly, by supplying the granule id of the desired granule.

The compressed key value in a sequence set entry represents the highest data record key that may be stored in the corresponding data control interval. In order to reduce index search time, a multilevel index is maintained. The index level immediately above the sequence set contains one index entry for each sequence set control interval. The entries are maintained in ascending key sequence. An index entry is not allowed to span an index control interval. Space is allocated within the index at the granularity of an index control interval. An index tree is maintained where the top level of the index tree consists of a single index control interval. The entries in this level of the index point down to index control intervals in the next lower level of the index, continuing to the lowest level of the index. The data records are stored within data control intervals. A sequence set index control interval exists for each data control area. It shows the key order of the data control intervals within the data control area and the set of free space control intervals within this data control area (if any). The index above the sequence set

forms a tree structure. The records within each data control interval are maintained in key sequence.

3.1.3 Data Layout Component

The data layout component of the data warehouse controls how the granules are positioned within the data storage component for optimal retrieval.

3.1.4 Data Access Component

The data access component of the data warehouse connects to a specified data source, analyzes the data, converts the data into granules and passes the granules on to the data layout component.

3.2 Data Storage Considerations

The granules are stored on disk in data pages. The data pages offer flexibility in accessing and processing the granules. Portions of the granules, called slices, are stored in the data pages, and the data pages are connected by links. The total storage cost for granules is based on the number of granules in the relation and the number of data pages per granule. The number of pages per granule is dependent upon the number of tuples in the relation and the number of tuples that can be stored per page. The equation for the total storage cost is the following:

*Storage cost = Number of granules * ⌈(Number of tuples / Max tuples per page)⌉ * Data page size*

For example, the relation has 1,000 granules and 1,000,000 tuples. Each data page is 4096 bytes, and 4080 out of 4096 bytes are available to store data for the granules. Then the following is the total storage costs:

*Storage cost = 1000 * ⌈(1,000,000 / (4080 * 8))⌉ * 4096*

= 125 Mbytes

3.3 Data Layout Considerations

One of the major operational expenses in a data mining application is the data retrieval cost. The way the data is laid out on the disk could have a significant impact on the time it takes to retrieve it. The granules are stored in CIs which are then grouped together in CAs. These need to be physically stored on a disk, which consists of cylinders and tracks. There are a number of issues to consider when mapping the CIs and the CAs on to a disk.

The benefits of having the CA allocations being done in cylinder sizes includes:

- It ensures that the dataset is on a cylinder boundary and does not span cylinders
- It reduces processing for I/O operations crossing track boundaries because "end-of-extent" does not have to be checked for each track.

- Large CAs consolidate indexes resulting in fewer index I/Os and levels, and ensure that there is 1 sequence set index record per CA so that large CAs will result in a smaller number of CAs and index records.
- CA free space will be used more efficiently.
- It helps the CI splits go faster as the heads are positioned over that cylinder.

The major disadvantage of allocating CAs in cylinders is as follows:

- CA splits will take longer.
- The usage of a small data CI size and a large CA size can cause an oversized index CI size, and the reverse can cause an underutilized index CI.

When allocating free space ensure that the record size plus the length of RDFs and CIDFs is taken into account. If there is an even distribution of inserts then it is best to specify the majority as CI free space and a little CA free space. If it is uneven then a little CI free space and a lot of CA free space. Free space is specified as Free space (a b), where "a" is the percentage of free space in each CI and "b" is the percent of free CIs in each CA. Use of free space reduces CI/CA splits if used properly. Insufficient or excessive free space can degrade performance and over utilize DASD. If there is insufficient free space, then CI/CA splits will degrade sequential processing as the dataset will not be stored in physical sequence. An excessive amount will cause DASD space to be wasted due to the reduction of the effective blocking factor of the data. Also, extra I/O will be required to transfer partially filled CIs in sequential processing.

Small data CI sizes reduce wasted data transfers as data warehouse always reads a CI. If processing is random or dynamic, a large CI is hopelessly inefficient. For files primarily accessed by CICS, the best data CI sizes are 2k or 4k. When using a large CISZ, I/O operations are reduced if multiples of 4096 are used. Also, note that on CKD, CIs are stored as physical records with inter record gaps, which should be taken into account when estimating response times. For direct access on-line files, a CISZ such as 2k or 4k should be used, and for sequential/mixed access a larger CISZ should be used.

Rotational delay is reduced as the read/write arm of the disk is placed above the sequence set and data. However, index updates cause an overhead as each copy of the sequence set has to be updated.

4 Data Mining: Finding Association Rules

An association rule is a rule that implies certain association relationships among a set of objects (such as "occur together" or "one implies the other") in a database. An association rule identifies a combination of attribute values or items that occur together with greater frequency than might be expected if the values or items were independent of one another. A common method to summarize the data is to find the association rules or patterns within that data. The task to find association rules may be difficult because the data may contain inaccuracies and inconsistencies or the data could be extremely large. Further more the number of patterns maybe incomprehensible. An association rule exists in a given relation if a certain percentage of data has that pattern. A potential pattern becomes an association rule if the intersection of the granules within that potential pattern results in a count that exceeds the minimum count of that relation. There are various ways of finding association rules. Here we focus on three granule-based algorithms to find the association rules.

4.1 Algorithms

In a data mining application the task of finding patterns within the data consists of two main steps. The first involves finding the set of all frequent itemsets. The second involves testing and generating all high confidence rules among these itemsets. The real benefit of using granules is the processing speed. Combining the process of performing a logical operation (OR, AND or NOT) on a series of granules is very efficient [2],

particularly compared with performing similar processes on lists of row-Ids. The operations performed on the granules are natural instructions for the computer. This has a great significance in the speed of computation. The intersection of the granules offers fast computation in finding association rules. The AND, SHIFT, and COUNT operations among granules are extremely fast as well. The use of granules improves the performance to find association rules [8].

4.1.1 Granule Based Lattice

The algorithm, Granule Based Lattice builds and populates a lattice from the relationship between the granules in the data. It then uses the lattice to generate the patterns within that data set.

This algorithm first finds all the granules that meet the minimum support level. These granules are referred to as level-one patterns. It then sorts the level-one patterns based on either increasing or decreasing order of their support levels. These granules are then intersected and based on the conditions specified below the lattice is populated. Once the lattice is built, the patterns are generated by analyzing the lattice.

The lattice is based on a forest data structure. It holds the relationship between different granules in the data. Each granule is represented by a node in the lattice. The nodes in the lattice can be easily traversed based on the relationship between different granules e.g. parent-child relationship. It is very efficient to directly access a node in the lattice and follow its relationship hierarchy. Here is high level look at the Granule Based Lattice algorithm:

1) Construct the granule for each unique attribute value.

2) Nodes = { I_i x $B(I_i)$: $|B(I_i)|$ > minsup }

3) Sort the item in Nodes based on some order, (e.g. increasing order of the bit count of the corresponding bitmap)

4) for each node I_i x $B(I_i)$ in Nodes

5) NewNode = θ and I = I_i

6) for each sibling I_j after I_i in the Nodes

7) I = $I_i \cup I_j$ and B-comb = $B(I_i) \cap B(I_j)$

8) // check the property of the new bitmap B-comb

9) if |B-combl > minsup then

   a) if $B(I_i) = B(I_j)$ then

      Remove I_j from Nodes

      Replace all I_i with I // I = $I_i \cup I_j$

   b) else if $B(I_i) \subset B(I_j)$ then

      Replace all I_i with I // I = $I_i \cup I_j$

   c) else if $B(I_i) \supset B(I_j)$ then

      Remove I_j from Nodes

      Add I x B-comb to the NewNode // I = $I_i \cup I_j$

   d) else if $B(I_i)$!= $B(I_j)$ then

      Add I x B-comb to the NewNode // I = $I_i \cup I_j$

10) if NewNode != θ then continue the expanding

11) C = C $\cup$ I // if I is not subsumed

Conditions:

Let's assume that we are processing the branch I_1 x B (I_1) and we want to combine it with its sibling 1_2 x B (I_2), that is $I_1 <= I_2$, (under a suitable total order). The main computation in Bit-Assoc relies on the following properties:

1. If $B(I_1) = B(I_2)$, then $B(I_1 \cup I_2) = B(I_1) \cap B(I_2) = B(I_1) = B(I_2)$.

   Thus we can simply replace every occurrence of I_1 with $I_1 \cup I_2$, and remove I_2 from further consideration, since its closure is identical to the closure of $I_1 \cup I_2$. In other words we treat $I_1 \cup I_2$ as a composite itemset.

2. If $B(I_1) \subset B(I_2)$, then $B(I_1 \cup I_2) = B(I_1) \cap B(I_2) = B(I_1)$!= $B(I_2)$.

   Here we can replace every occurrence of I_1 with $I_1 \cup I_2$, since if I_1 occurs in any transaction, then I_2 always occurs there too. But since $B(I_1)$!= $B(I_2)$, we cannot remove I_2 from consideration, it generates a different closure.

3. If $B(I_1) \supset B(I_2)$. then $B(I_1 \cup I_2) = B(I_1) \cap B(I_2) = B(I_2)$!= B (I_1).

   Here we can replace every occurrence of I_2 with $I_1 \cup I_2$, since if I_2 occurs in any transaction, then I_1 always occurs there too. I_1 however, produces a different closure and it must be retained.

4. If $B(I_1)$!= $B(I_2)$, then $B(I_1 \cup I_2)$=$B(I_1) \cap B(I_2)$!= $B(I_1)$!= $B(I_2)$.

In this case, nothing can be eliminated; both I_1 and I_2 lead to different closure. The basic algorithm to find the association rules in the given data set using Granule Based Lattice is as follows:

1) Retrieve all the granules.
2) Find the granules that meet the minimum support level (Level One patterns).
3) Sort them based on some order (e.g. ascending order of their support level).
4) Build the lattice.
5) Generate the patterns from the lattice.

The flow chart of this algorithm is pictured below:

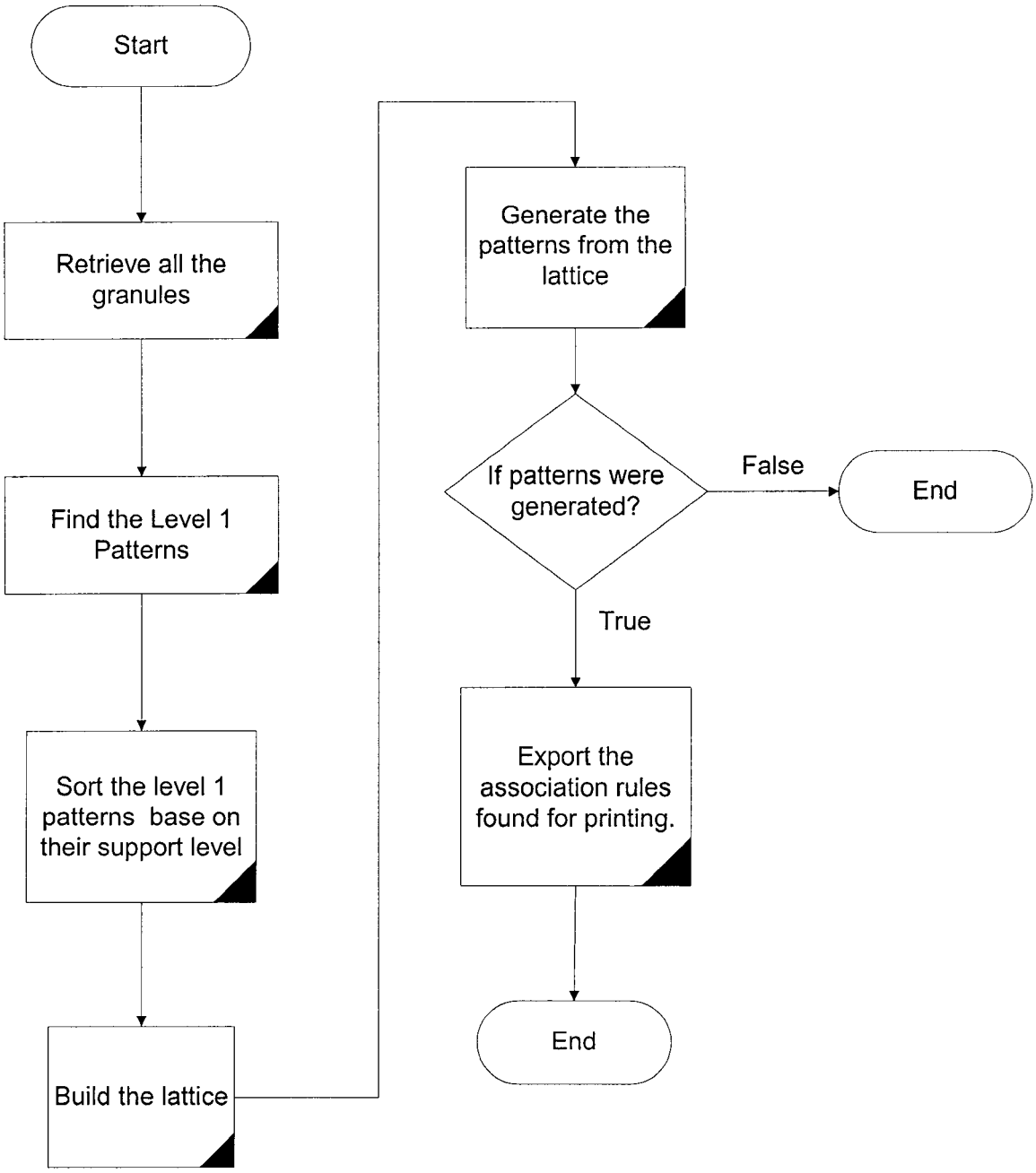


Figure 5. Flow chart to find association rules using Granule Based Lattice

It is widely recognized that the set of association rules can rapidly grow to be unwieldy, especially as we lower the frequent requirements. The larger the set of frequent itemsets the more the number of rules presented to the user, many of which are redundant. This is true even for sparse datasets, but for dense datasets it is simply not feasible to mine all frequent itemsets, let alone to generate the rules between the itemsets. The Granule Based Lattice algorithm is tuned to produce non-redundant rules.

4.1.2 Granule Based Appriori

The algorithm, Granule Based Appriori, is based on the Appriori algorithm which was developed by Rakesh Agrawal. It generates new potential patterns based on the previous level's patterns [3].

Granule Based Appriori employs a level-wise search approach where patterns of size S (S-itemsets), or in short S-patterns, are used to find potential patterns of size (S + 1). The set of S-patterns where S = 1 are known as Level-1 patterns. These are denoted by L1 and are used to find L2, the set of Level-2 patterns, which in turn are used to find L3, and so on, until no more patterns can be found. At each level, potential patterns of length S, or in short S-potential patterns, are generated based on (S-1)-patterns of the previous level. In other words, when generating a potential pattern of size S, one verifies that all its (S – 1) sub-patterns are association rules. If an (S – 1) sub-pattern does not exist as an association rule of (S – 1) level then this candidate is removed from the list of potential patterns. Since the size of these granules may be large, slices of each granule are read and processed at a time. In the end, if the intersection of granules in the potential

pattern results in a count that meets or exceeds the minimum support level, the potential pattern is an association rule, and it is saved. The cycle stops when no S-potential patterns are found to be association rules or if no new (S + 1)-potential patterns can be generated from the discovered patterns of size S.

Initially all the granules that meet the minimum support level are found. These level-1 patterns are then sorted based on either increasing or decreasing order of their support levels. Level-2 patterns are then generated based on level-1 patterns. This algorithm continues generating potential patterns for the next levels based on the patterns generated for the previous levels. When no more potential patterns could be generated for a particular level, the algorithm stops.

Minimizing the number of potential patterns generated is important to the performance of this algorithm. One of the factors that is considered to reduce the number of potential patterns is that the granules that belong to the same column can not form a pattern since the bit count of their resulting intersection is zero.

The basic algorithm to find the association rules in the given data set using Granule Based Appriori is as follows:

1) Retrieve all the granules.
2) Find the granules that meet the minimum support level (Level One patterns).
3) Sort them based on some order (e.g. ascending order of their support level).
4) Move to the next level.
5) Generate the potential patterns for the current level based on the patterns found for the previous level.

6) Perform the required intersections to verify that the potential pattern is indeed a pattern.
7) Continue to step 4 and then repeat steps following that until no more potential patterns can be generated.

The flow chart of this algorithm is pictured below:

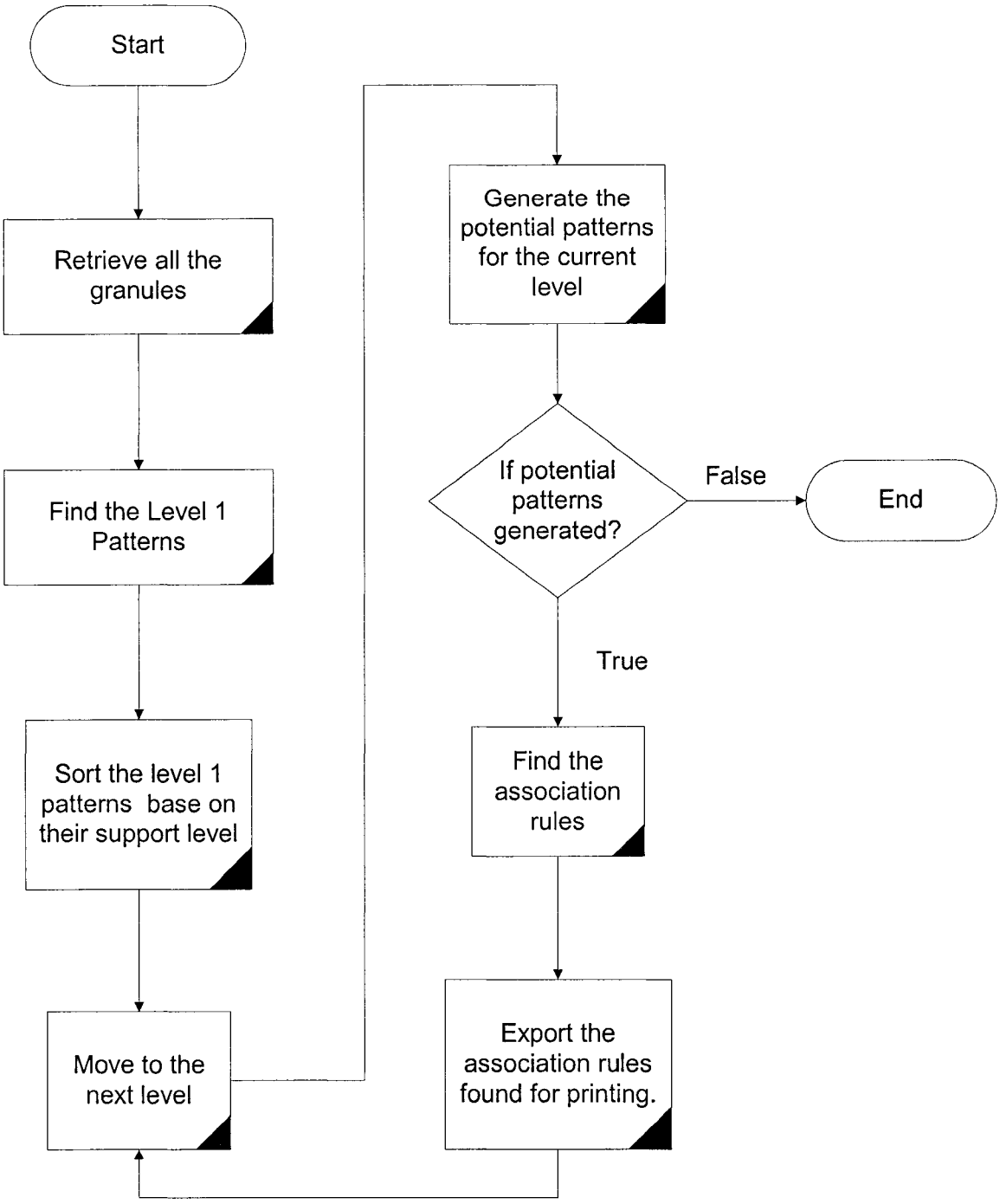


Figure 6. Flow chart to find association rules using Granule Based Appriori

4.1.3 Granule Based Appriori Lattice

The algorithm, Granule Based Appriori Lattice is partially based on Granule Based Appriori and Granule Based Lattice algorithms. Just like Granule Based Appriori algorithm, it generates new potential patterns based on the previous level's patterns. It also creates and uses a lattice to reduce to a minimum the number of intersections required to verify that a potential pattern is indeed a pattern.

This algorithm first finds all the granules that meet the minimum support level. These granules are referred to as level-1 patterns. It then sorts the level-1 patterns based on either increasing or decreasing order of their support levels. Level-2 patterns are then generated based on level-1 patterns. This algorithm continues generating potential patterns for the next levels based on the patterns generated for the previous levels. When no more potential patterns could be generated for a particular level, the algorithm stops.

The lattice holds the relationship between all the granules that are examined. As the granules are intersected the lattice is updated based on the following conditions. Let's assume that we are processing the branch I_1 x B (I_1) and we want to combine it with its sibling 1_2 x B (I_2), that is $I_1 <= I_2$, (under a suitable total order). The main computation in Bit-Assoc relies on the following properties:

1. If $B(I_1) = B(I_2)$, then $B(I_1 \cup I_2) = B(I_1) \cap B(I_2) = B(I_1) = B(I_2)$.

   Thus, we can simply replace every occurrence of I_1 with $I_1 \cup I_2$, and remove I_2 from further consideration, since its closure is identical to the closure of $I_1 \cup I_2$. In other words we treat $I_1 \cup I_2$ as a composite itemset.

2. If $B(I_1) \subset B(I_2)$, then $B(I_1 \cup I_2) = B(I_1) \cap B(I_2) = B(I_1) != B(I_2)$.

Here we can replace every occurrence of I_1 with $I_1 \cup I_2$, since if I_1 occurs in any transaction, then I_2 always occurs there too. But since $B(I_1) != B(I_2)$, we cannot remove I_2 from consideration, it generates a different closure.

3. If $B(I_1) \supset B(I_2)$. then $B(I_1 \cup I_2) = B(I_1) \cap B(I_2) = B(I_2) != B(I_1)$.

Here we can replace every occurrence of I_2 with $I_1 \cup I_2$, since if I_2 occurs in any transaction, then I_1 always occurs there too. I_1 however, produces a different closure and it must be retained.

4. If $B(I_1) != B(I_2)$, then $B(I_1 \cup I_2)=B(I_1) \cap B(I_2) != B(I_1) != B(I_2)$.

In this case, nothing can be eliminated; both I_1 and I_2 lead to different closure.

Minimizing the number of potential patterns generated is important to the performance of this algorithm. Each potential pattern generated must have the possibility of being declared an association rule, because the process of verifying that each of these potential patterns is an association rule is computationally and resource wise expensive.

To reduce the number of potential patterns generated at each level and to avoid the unnecessary and costly intersections a variety of factors are considered. The granules that belong to the same column are not intersected since their intersection is an empty set and therefore they do not form a pattern. The Appriori property described below is also used to reduce the number of potential patterns generated. So for a potential pattern to be an association rule all its sub-patterns need to be association rules. This property is based

on the following observation. By definition, if a granule G does not satisfy the minimum support level threshold Min_Support, then G is not an association rule, that is, Support (G) < Min_Support. If the granule G is part of a potential pattern P then P can not occur more frequently than G. Therefore P would not be an association rule either. This property belongs to a special category of properties called anti-monotone in a sense that if a set cannot pass a test, all of its supersets will fail the same test as well.

When the granules in a potential pattern need to be intersected to verify that the potential pattern is an association rule the lattice is used to avoid the unnecessary and costly intersections. Thus in this algorithm not only the number of potential patterns generated is reduced but also the number of intersections required to verify those patterns are reduced.

The basic algorithm to find the association rules in the given data set using Granule Based Appriori Lattice is as follows:

1) Retrieve all the granules.
2) Find the granules that meet the minimum support level (Level One patterns).
3) Sort them based on some order (e.g. ascending order of their support level).
4) Move to the next level.
5) Generate the potential patterns for the current level based on the patterns found for the previous level.
6) Use the lattice to reduce to a minimum the number of intersections required to verify that a potential pattern is indeed a pattern.

7) Perform the required intersections to verify that the potential pattern is indeed a pattern.
8) Update the lattice based on the relationship between the granules that were intersected.
9) Continue to step 4 and then repeat steps following that until no more potential patterns can be generated.

The flow chart of this algorithm is pictured below:

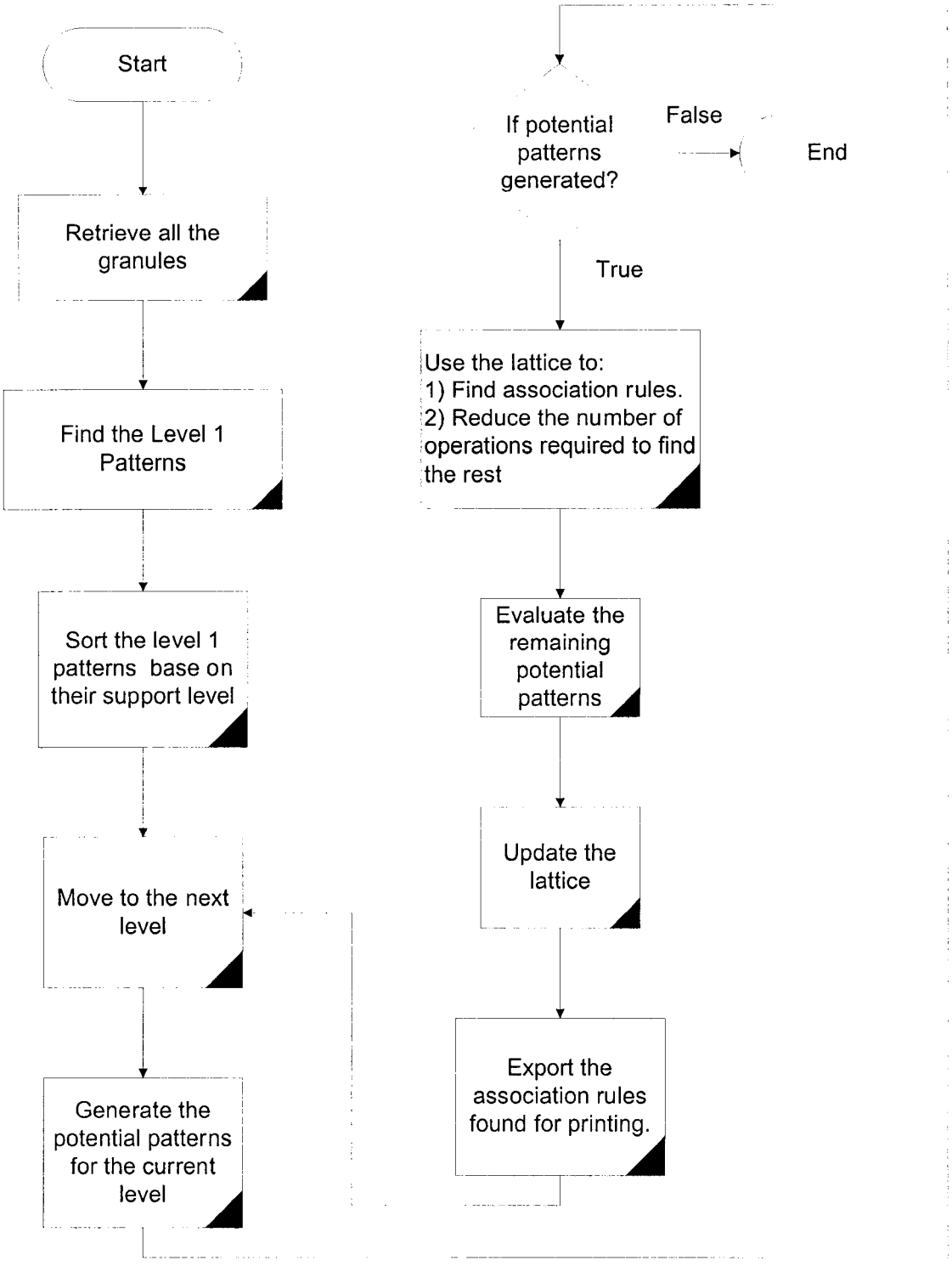


Figure 7. Flow chart to find association rules using Granule Based Appriori Lattice

4.2 Computation Cost

When finding association rules, the granules in a potential pattern are intersected and their result is counted. Since the granules are represented in a bit form, the intersection involves bit-wise AND operations. The bit-wise AND operations are done using the machine's word size instructions. This is a costly operation and that is why it is so important not to generate all the possible combinations of the granules. Thus the number of potential patterns generated is an important factor to consider in each of these algorithms.

The number of the AND operations on a k-candidate potential pattern is

$\lceil$ Number of bits per granule / (Number of bits per word) $\rceil$ * (k –1)

For example, a 4-candidate potential pattern consists of 4 granules. If the relation has 20480 tuples, then there are 640 words (20480 / 32) in a granule. Between four granules, there are 3 AND operations required. Thus, a total of 1920 AND operations (20480 / 32 * 3 = 1920) are needed to perform the intersection of the four granules.

After the AND operations are completed, the bits in the result need to be counted to determine if the potential pattern is an association rule. It is crucial to have a fast algorithm to count the bits in the resulted granule because this has a huge impact on the performance of the algorithm.

5 Comparison

The algorithms, Granule Based Lattice, Granule Based Appriori and Granule Based Appriori Lattice are compared based on the number of intersections they perform. For each of these algorithms the number of intersections performed determines the cost of the algorithm. The larger the number of intersections, the more expensive the algorithm. A future enhancement would be to accurately measure the process time for each of the above specified algorithms.

5.1 Data Specification

It has been observed that the performance of the above specified algorithms could be significantly affected by the extent of the hierarchies within the data. A hierarchy is defined as a subset or a superset relationship between the granules within the data. Due to this observation the comparisons between these algorithms are based on two different types of data, flat types and hierarchical types of data.

5.1.1 Flat Data

A flat type of data contains a minimal number of hierarchies. The characteristics of each flat dataset that is used in the comparisons are described below. These datasets differ only in their number of tuples. The table size specifies the size of the database table

while the granule size specifies the size of the same data converted to granules and stored in the data warehouse.

Dataset	Number of Rows	Number of Columns	Table Size	Granule Size
1	100	12	0.14 MB	2 KB
2	1,000	12	1.4 MB	15 KB
3	10,000	12	14.3 MB	146 KB
4	100,000	12	143.2 MB	1.5 MB
5	1,000,000	12	1,432.3 MB	14.5 MB

Table 2. Data specification for flat data type

Each column of the data in the database has a certain number of unique attributes and these remain the same for each dataset.

Column Id	Column Name	Number of Unique Attributes	Dependency
1	Make	4	x
2	Model	4	1
3	ExteriorColor	20	x
4	InteriorColor	10	x
5	Year	10	x
6	OwnerRace	10	x
7	Transmission	4	x
8	Features	10	x
9	Seats	10	x
10	Roof	10	x
11	Entertainment	10	x
12	Navigation	5	x

Table 3. Attribute value specification for flat data type

5.1.2 Hierarchical Data

A hierarchical type of data contains a much larger number of hierarchies than the flat type of data. The characteristics of each hierarchical dataset that is used in the comparisons are described below. These datasets differ only in their number of tuples. The table size specifies the size of the database table while the granule size specifies the size of the same data converted to granules and stored in the data warehouse.

Dataset	Number of Rows	Number of Columns	Table Size	Granule Size
1	100	12	0.14 MB	4 KB
2	1,000	12	1.4 MB	39 KB
3	10,000	12	14.3 MB	386 KB
4	100,000	12	143.2 MB	3.9 MB
5	1,000,000	12	1,432.3 MB	38.6 MB

Table 4. Data specification for hierarchical data type

Each column of the data in the database has a certain number of unique attributes and these remain the same for each dataset.

Column Id	Column Name	Number of Unique Attributes	Dependency (Column Id)
1	C1	5	x
2	C2	25	1
3	C3	125	2
4	C4	20	x
5	C5	10	x

6	C6	5	x
7	C7	2	x
8	C8	4	7
9	C9	8	8
10	C10	16	9
11	C11	32	10
12	C12	64	11

Table 5. Attribute value specification for hierarchical data type

5.2 System Information

Machine	IBM ThinkPad T30
Processor	Intel Pentium 4, 2.0 GHz
Memory	1 GB
Drive	60 GB
Operating System	Windows 2000

Table 6. System information

5.3 Performance Evaluation

The following information is gathered by running the algorithms against the datasets specified above. The performance of each algorithm is measured based on the number of intersections performed.

5.3.1 Flat Data

Here the ratio between the data and the support level is kept constant.

Data Set 1

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	10	56	2	1230
Granule Based Appriori	10	56	2	1176
Granule Based AppriorLattice	10	56	2	1176

Table 7. Performance numbers for flat data set 1

Data Set 2

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	100	49	1	1078
Granule Based Appriori	100	49	1	1078
Granule Based AppriorLattice	100	49	1	1078

Table 8. Performance numbers for flat data set 2

Data Set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	1000	47	1	991
Granule Based Appriori	1000	47	1	991
Granule Based	1000	47	1	991

ApprioriLattice				

Table 9. Performance numbers for flat data set 3

Data Set 4

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	10000	48	1	1034
Granule Based Appriori	10000	48	1	1034
Granule Based ApprioriLattice	10000	48	1	1034

Table 10. Performance numbers for flat data set 4

Data Set 5

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	100000	48	1	1034
Granule Based Appriori	100000	48	1	1034
Granule Based ApprioriLattice	100000	48	1	1034

Table 11. Performance numbers for flat data set 5

Here the dataset is kept constant while changing the minimum support level.

Data Set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	31	14767	4	71611336
Granule Based Appriori	31	14767	4	400942
Granule Based AppriorilLattice	31	14767	4	399329

Table 12. Performance numbers for flat data set 3

Data Set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	62	5892	3	5034464
Granule Based Appriori	62	5892	3	177680
Granule Based AppriorilLattice	62	5892	3	176929

Table 13. Performance numbers for flat data set 3

Data Set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	125	1450	3	361609
Granule Based Appriori	125	1461	3	15674
Granule Based AppriorilLattice	125	1461	3	15566

Table 14. Performance numbers for flat data set 3

Data Set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	250	473	2	48864
Granule Based Appriori	250	473	2	7216
Granule Based ApprioriLattice	250	473	2	7216

Table 15. Performance numbers for flat data set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	500	162	2	6651
Granule Based Appriori	500	162	2	5519
Granule Based ApprioriLattice	500	162	2	5519

Table 16. Performance numbers for flat data set 3

Data Set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	1000	47	1	991
Granule Based Appriori	1000	47	1	991
Granule Based	1000	47	1	991

ApprioriLattice				

Table 17. Performance numbers for flat data set 3

5.3.2 Hierarchical Data

Here the ratio between the data and the support level is kept constant.

Data Set 1

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	10	74	4	661
Granule Based Appriori	10	74	4	381
Granule Based ApprioriLattice	10	74	4	357

Table 18. Performance numbers for hierarchical data set 1

Data Set 2

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	100	69	3	667
Granule Based Appriori	100	69	3	357
Granule Based ApprioriLattice	100	69	3	343

Table 19. Performance numbers for hierarchical data set 2

Data Set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	1000	69	3	665
Granule Based Appriori	1000	69	3	357
Granule Based AppriorLattice	1000	69	3	343

Table 20. Performance numbers for hierarchical data set 3

Data Set 4

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	10000	67	3	615
Granule Based Appriori	10000	67	3	357
Granule Based AppriorLattice	10000	67	3	343

Table 21. Performance numbers for hierarchical data set 4

Data Set 5

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	100000	68	3	655
Granule Based	100000	68	3	333

Appriori				
Granule Based AppriorLattice	100000	68	3	319

Table 22. Performance numbers for hierarchical data set 5

Here the dataset is kept constant while changing the minimum support level.

Data Set 3

Algorithm	**Minimum Support Level**	**Number of Patterns Found**	**Longest Pattern Length**	**Number of Intersections Performed**
Granule Based Lattice	62	11276	6	82086729
Granule Based Appriori	62	11276	6	75545
Granule Based AppriorLattice	62	11276	6	58748

Table 23. Performance numbers for hierarchical data set 3

Data Set 3

Algorithm	**Minimum Support Level**	**Number of Patterns Found**	**Longest Pattern Length**	**Number of Intersections Performed**
Granule Based Lattice	125	4522	6	3342064
Granule Based Appriori	125	4522	6	26798
Granule Based AppriorLattice	125	4522	6	19411

Table 24. Performance numbers for hierarchical data set 3

Data Set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	250	1142	5	156313
Granule Based Appriori	250	1142	5	8652
Granule Based AppriorLattice	250	1142	5	7392

Table 25. Performance numbers for hierarchical data set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	500	274	4	9743
Granule Based Appriori	500	274	4	1755
Granule Based AppriorLattice	500	274	4	1581

Table 26. Performance numbers for hierarchical data set 3

Data Set 3

Algorithm	Minimum Support Level	Number of Patterns Found	Longest Pattern Length	Number of Intersections Performed
Granule Based Lattice	1000	69	3	665
Granule Based Appriori	1000	69	3	357
Granule Based AppriorLattice	1000	69	3	343

Table 27. Performance numbers for hierarchical data set 3

5.4 Observations and Explanations

Here are some observations and explanations on the results.

5.4.1 General

1) Based on the information in the Tables 2 and 4, it is observed that the size of the granules stored in the data warehouse is much smaller than the size of the records stored in the database. The size of the column values, the number of unique attribute values and the number of records in the database are among the main factors affecting this ratio.
2) Granule Based Appriori Lattice, Granule Based Lattice and Granule Based Appriori algorithms all use the data warehouse to mine the data. This implies that the time taken to convert the data in the database into granules is the same for the three algorithms. This would also mean that the performance of these algorithms is improved substantially by avoiding time-consuming table scans.
3) The three granule based algorithms specified above use the same routine to find the first level association rules. This implies that they all take the same amount of time finding the first level patterns.
4) They also use a similar routine to find the second level association rules. This implies that they also take the same amount of time to find the second level patterns.

5) Granule Based Lattice algorithm is suited to find the longest patterns in the data. On the other hand Granule Based Appriori Lattice and Granule Based Appriori algorithms are suited to find all the patterns in the data. This implies that an algorithm would incur more cost in execution time to extract the patterns in the other form.

5.4.2 Flat Data

6) Based on the information in the Tables 7 through 11, it is observed that in the case the data is not hierarchical in nature the number of the intersections performed by Granule Based Appriori Lattice algorithm is the same as the number of the intersections performed by Granule Based Appriori algorithm.
7) Based on the information in the Tables 12 through 17, it is observed that in the case the data is even slightly hierarchical in nature and those hierarchies qualify as patterns, the number of the intersections performed by Granule Based Appriori Lattice algorithm decreases compared to those of Granule Based Appriori algorithm.
8) Based on the information in the Tables 7 through 11, it is observed that in the case the data is not hierarchical in nature and the length of the patterns found is quite small, the number of the intersections performed by Granule Based Lattice algorithm follows closely the number of intersections performed by Granule Based Appriori Lattice and Granule Based Appriori algorithms.
9) Based on the information in the Tables 12 through 17, it is observed that, as the number and the length of the patterns found increases, the number of the

intersections performed by Granule Based Lattice algorithm increases rapidly as compared to those of Granule Based Appriori Lattice and Granule Based Appriori algorithms.

5.4.3 Hierarchical Data

10) Based on the information in the Tables 18 through 22, it is observed that in the case the data is hierarchical in nature the number of the intersections performed by Granule Based Appriori Lattice algorithm is smaller than the number of intersections performed by Granule Based Appriori algorithm.

11) Based on the information in the Tables 23 through 27, it is observed that in the case the data is hierarchical in nature and those hierarchies qualify as patterns, the number of the intersections performed by Granule Based Appriori Lattice algorithm decreases substantially compared to those of Granule Based Appriori algorithm.

12) Based on the information in the Tables 23 through 27, it is also observed that as the hierarchies get deeper and as the length of the patterns increases, the number of the intersections performed by Granule Based Appriori Lattice algorithm decreases substantially compared to those of Granule Based Appriori algorithm.

13) Comparing the information gathered from runs made against the flat data sets and those made against the hierarchical datasets, we observe that the Granule Based Appriori Lattice algorithm at worst case performs as many intersections as Granule Based Appriori algorithm but as the data becomes more

hierarchical number of intersections it performs decreases significantly as compared to Granule Based Appriori algorithm.

14) Based on the performance runs made against the hierarchical data sets (Data Sets 1 through 5) and data set (Data Set 3) with different support levels, it is observed that the number of intersections performed by the Granule Based Lattice algorithm is much higher than that of the Granule Based Appriori Lattice and Granule Based Appriori algorithms. This deterioration could be attributed to the huge number of intersections required to find the relationship between a qualified granule and the granules that have already populated the lattice.

15) Based on the information in the Tables 18 through 22, it is observed that in the case of hierarchical data, even if the number and the length of the patterns found is small, the number of the intersections performed by Granule Based Lattice algorithm is substantially higher than the number of intersections performed by Granule Based Appriori Lattice and Granule Based Appriori algorithms.

16) Based on the information in the Tables 23 through 27, it is observed that, as the number and the length of the patterns found increases, the number of the intersections performed by Granule Based Lattice algorithm increases almost exponentially as compared to those of Granule Based Appriori Lattice and Granule Based Appriori algorithms.

6 Summary

The bit representation of the granules offers efficient storage; the granule sizes remain small and the entire database can be fully converted to granules and made available for ad hoc queries in less space than it takes to store the raw data on most databases. Tuning is data-dependent, allowing data to be optimized once for any number of ad hoc queries. Because granules are compact, more data can be kept in memory for subsequent queries, there by speeding throughput on iterative analysis. Queries are resolved by efficiently combining and manipulating granules on only the relevant columns. This avoids time-consuming table scans. The disadvantage of using the granules is the cost of creating them. However, once the data is converted into a compact form, it can benefit many different data mining queries.

Our experiments show that the performance based on the number of intersections done by the newly proposed Granule Based Appriori Lattice algorithm at worst case is as good as or better than the performance of Granule Based Appriori and Granule Based Lattice algorithms and at best improves significantly compared to the performance of the latter two algorithms.

7 Manual

This section describes the different applications implemented for this thesis.

7.1 Data Generator Application

This application is used to generate the required data set. It has the capability to load the data into a database if specified. It can create the table if it doesn't exist.

7.1.1 Usage

java -Xms512m -Xmx512m -classpath

..\Lib\Thesis.jar;..\Lib\xercesImpl.jar;..\Lib\xercesSamples.jar;..\Lib\xml-

apis.jar;..\Lib\xmlParserAPIs.jar;..\Lib\XMLWrapper.jar

com.datagenerator.DataGenerator -n..\config\DataGeneratorConfig.xml

7.1.2 Configuration

Sample configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<DataGeneratorConfigFile>
  <GeneralSettings>
    <!-- number of records that need to be generated -->
    <TotalNumRecords><![CDATA[10]]></TotalNumRecords>

    <!-- number of records per batch/file -->
    <NumRecordsPerBatch><![CDATA[10]]></NumRecordsPerBatch>

    <!-- Full path to the Output file with no extentions -->
    <OutputFileName><![CDATA[c:\tmp\DataGenerated]]></OutputFileName>
```

```
    <!-- Delimeter to enclose the values -->
    <Delimeter><![CDATA[]]></Delimeter>

    <!-- whether to generated the ctrl file -->
    <GenerateCtrlFile><![CDATA[false]]></GenerateCtrlFile>

    <!-- Full path to the control file name with no extentions -->
    <CtrlFileName><![CDATA[c:\tmp\CtrlFile]]></CtrlFileName>

    <!-- Table to insert the data into -->
    <TableName><![CDATA[Car]]></TableName>

    <!-- Need to load the data using sql loader -->
    <LoadData><![CDATA[false]]></LoadData>

    <UserName><![CDATA[db2admin]]></UserName>
    <Password><![CDATA[db2admin]]></Password>
    <Database><![CDATA[TestDB]]></Database>
  </GeneralSettings>
    <ColumnSection>
      <!-- If unique is set to true, then the Value field should have just one value -->
      <!-- the type should be specified then and if the type is string the value is  -->
      <!-- used as the base and a unique number is going to be appended to it starting --
>
      <!-- from 1, in case the type is an int it is going to be incremented by 1 -->
      <!-- Valid types at this point are "string" and "int" -->
        <ColumnName name="OwnerId" unique="true" type="string" inuse="true">
        <!-- percentage is the percent of the time this value is going to be picked and --
>
        <!-- they should always add up to 100 percent for a ColumnName -->
            <Value>
              <Data>
                <![CDATA[Owner]]>
              </Data>
            </Value>
        </ColumnName>
        <ColumnName name="Made" unique="false" type="string" inuse="true">
        <!-- percentage is the percent of the time this value is going to be picked and --
>
        <!-- they should always add up to 100 percent for a ColumnName -->
            <Value>
              <Percentage>25</Percentage>
              <Data>
```

```
                    <![CDATA[Honda]]>
                  </Data>
                </Value>
                <Value>
                  <Percentage>25</Percentage>
                  <Data>
                    <![CDATA[Toyota]]>
                  </Data>
                </Value>
                <Value>
                  <Percentage>25</Percentage>
                  <Data>
                    <![CDATA[Nissan]]>
                  </Data>
                </Value>
                <Value>
                  <Percentage>25</Percentage>
                  <Data>
                    <![CDATA[BMW]]>
                  </Data>
                </Value>
            </ColumnName>
        </ColumnSection>
        <Dependency>
            <Column>
                <ColumnName><![CDATA[Model]]></ColumnName>
                <DependsOn>
                    <ColumnName><![CDATA[Made]]></ColumnName>
                    <DepType><![CDATA[NotUsedYet]]></DepType>
                </DependsOn>
            </Column>
        </Dependency>
</DataGeneratorConfigFile>
```

7.2 Data Warehousing Application

This application is used to access a database, read its records, generate granules and lay them out on disk based on the layout algorithm.

7.2.1 Usage

java -Xms512m -Xmx800m -classpath
..\Lib\Thesis.jar;..\Lib\xercesImpl.jar;..\Lib\xercesSamples.jar;..\Lib\xml-apis.jar;..\Lib\xmlParserAPIs.jar;..\Lib\XMLWrapper.jar;..\Lib\db2java.zip
com.datawarehouse.DataWarehouse

7.2.2 Configuration

Sample configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All the parameters here are explained in the java doc for Data Warehouse class. -
->
<DataWareHouseConfigFile>
  <GeneralSettings>

    <!-- Generally configure, TableName, VsamFileName and RESTORE_DIR. -->

    <!-- Database related information -->

<JDBC_DRIVER><![CDATA[COM.ibm.db2.jdbc.app.DB2Driver]]></JDBC_DRI
VER>
    <URL><![CDATA[jdbc:db2:TESTDB]]></URL>
    <UserName><![CDATA[db2admin]]></UserName>
    <Password><![CDATA[db2admin]]></Password>
    <TableName><![CDATA[hmoobed.cars_10000]]></TableName>
    <ColNames><![CDATA[Make, Model, ExteriorColor, InteriorColor, Year,
OwnerRace, Transmission, Features, Seats, Roof, Entertainment,
Navigation]]></ColNames>

    <!-- Disk related information -->
    <Latency><![CDATA[10]]></Latency>
    <SeekTime><![CDATA[20]]></SeekTime>
    <TransferTime><![CDATA[30]]></TransferTime>
    <NumDisks><![CDATA[1]]></NumDisks>

    <!-- Vsam related information -->
    <KeySize><![CDATA[2]]></KeySize>
    <PageSize><![CDATA[1024]]></PageSize>
    <ControlSize><![CDATA[4]]></ControlSize>
    <ContIntSize><![CDATA[1024]]></ContIntSize>
    <NumContIntPerTrack><![CDATA[16]]></NumContIntPerTrack>
    <NumTracksPerContArea><![CDATA[16]]></NumTracksPerContArea>
```

```
        <NumContAreasPerDisk><![CDATA[16]]></NumContAreasPerDisk>

    <VsamFileName><![CDATA[C:\HoomanDataFolder\Data\10000\Disk]]></VsamFil
    eName>

        <!-- Data Layout related information -->
        <GranSecSizeInBytes><![CDATA[2]]></GranSecSizeInBytes>

        <!-- Other settings -->

    <RESTORE_DIR><![CDATA[C:\HoomanDataFolder\Restore\10000]]></RESTOR
    E_DIR>

    <COL_NAME_ATTR_VALUE_SEPARATOR><![CDATA[__]]></COL_NAME_
    ATTR_VALUE_SEPARATOR>

      </GeneralSettings>
    </DataWareHouseConfigFile>
```

7.3 Data Mining Application

This application is used to run different data mining applications that are implemented.

7.3.1 Usage

```
java  -Xms512m -Xmx800m -classpath
c:\HoomanDataFolder\Lib\Thesis.jar;c:\HoomanDataFolder\Lib\xercesImpl.jar;c:\Hooma
nDataFolder\Lib\xercesSamples.jar;c:\HoomanDataFolder\Lib\xml-
apis.jar;c:\HoomanDataFolder\Lib\xmlParserAPIs.jar;c:\HoomanDataFolder\Lib\XMLW
rapper.jar com.dataminingapp.BenchMark true true true true
```

7.3.2 Configuration

Sample configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!-- All the parameters here are explained in the java doc for Data Warehouse class. -->
<DataMiningConfigFile>
  <GeneralSettings>

        <!-- Generally configure, MINIMUM_SUPPORT_LEVEL and OUTPUT_DIR.
-->

    <!-- Datamining related information -->

<MINIMUM_SUPPORT_LEVEL><![CDATA[31]]></MINIMUM_SUPPORT_LEVEL
>
    <PATTERN_DELIMETER><![CDATA[,]]></PATTERN_DELIMETER>

<OUTPUT_DIR><![CDATA[C:\HoomanDataFolder\Output\10000]]></OUTPUT_DIR
>
  </GeneralSettings>
</DataMiningConfigFile>
```

7.4 Benchmark Application

This application runs all three of the above specified algorithms and reports the time each one took to process the data.

7.4.1 Usage

```
java  -Xms512m -Xmx800m -classpath
c:\HoomanDataFolder\Lib\Thesis.jar;c:\HoomanDataFolder\Lib\xercesImpl.jar;c:\Hooma
nDataFolder\Lib\xercesSamples.jar;c:\HoomanDataFolder\Lib\xml-
apis.jar;c:\HoomanDataFolder\Lib\xmlParserAPIs.jar;c:\HoomanDataFolder\Lib\XMLW
rapper.jar com.dataminingapp.DataMiningApp
```

7.4.2 Configuration

Refer to the configuration parameters for the data warehouse and data mining sections.

7.5 Data Differentiator Application:

This application compares the outputs of two of the specified algorithms and creates two new diff files based on the original files with only the lines (patterns) missing from the other file.

7.5.1 Usage

```
java  -Xms512m -Xmx800m -classpath ..\Lib\Thesis.jar
com.datadifferentiator.DataDifferentiator ..\Output\AppriorиOutput.out
..\Output\AppriorиLatticeOutput.out
```

7.5.2 Configuration

None.

Works Cited

[1] Jiawei Han and Micheline Kamber, "Data Mining: Concepts and Techniques," Morgan Kaufmann. 2001.

[2] E. Louie and T. Y. Lin, "Finding Association Rules using Fast Bit Computation: Machine-Oriented Modeling," In: Foundations of Intelligent Systems, Z. Ras and S. Ohsuga (eds), Lecture Notes in Artificial Intelligence #1932, Springer-Verlag, 2000, pp. 486- 494.

[3] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami, "Mining association rules between sets of items in large databases," In: Proceedings of the ACM SIGMOD Conference on Management of Data, Washington, D.C., May, 1993. pp. 207-216

[4] Xiaohua Hu, T. Y. Lin and E. Louie, "Bitmap Techniques for Optimizing Decision Support Queries and Association Rule Algorithms," In: Exploring New Frontiers on Artificial Intelligence, Lecture Notes on Artificial Intelligent series, to appear

[5] E. Louie and T. Y. Lin, "Association Rules with Additional Semantics Modeled by Binary Relations," In: Rough Set Theory and Granular Computing, physica-Verlag, Shusaku Tsumoto, Masahiro Inuiguchi and Shoji Hirano (Eds)

[6] T. Y. Lin and E. Louie, "Modeling the Real World for Data Mining: Granular Computing Approach," In: Proceeding of Joint 9th IFSA World Congress and 20th NAFIPS International Conference, Vancouver , Canada, July 25-28, 2001, pp.3044-3049.

[7] I-Jen Chiang and T. Y. Lin, "Index Miner," In: Proceedings of the 25th International Conference on Computer Software and Applications, Chicago, October 8-12, 2001. pp. 613-614

[8] E. Louie, "Using Granules To Find Association Rules," In: Thesis Presented to The Faculty of the Department of Mathematics and Computer Science San Jose State University, San Jose, December, 2000. pp. 47